




Universität Hamburg

DER FORSCHUNG | DER LEHRE | DER BILDUNG

# Detecting Usage of Deprecated Web APIs via Tracing

SE 2025

Leif Bonorden & André van Hoorn

 The method `isSpace(char)` from the type `Character` is deprecated

```
boolean result = Character.isSpace('@');
```

*We know about Deprecation for this type of API.*

*But what about such APIs?*

```
GET https://www.character.org/isSpace?character= %40
```

# Detecting Usage of Deprecated Web APIs via Tracing

Bonorden & van Hoorn,  
ICSA 2024



## Detecting Usage of Deprecated Web APIs via Tracing

Leif Bonorden  
Universität Hamburg  
Hamburg, Germany  
leif.bonorden@uni-hamburg.de  
ORCID: 0000-0002-2131-7790

André van Hoorn  
Universität Hamburg  
Hamburg, Germany  
andre.van.hoorn@uni-hamburg.de  
ORCID: 0000-0003-2567-6077

**Abstract**—Deprecation is a way to inform clients using an application programming interface (API) that the usage of this API is discouraged. Tool support and research for deprecation in local APIs are well established. However, nowadays web APIs are more commonly used, e.g., using the REST architectural style. However, the techniques to detect and handle the usage of deprecated local APIs cannot be directly applied to web APIs. Previous approaches for detecting deprecated web APIs focus on static analysis of client code by detecting calls to web APIs and, subsequently, an investigation of associated API specifications. These approaches currently have two essential limitations: (i) The target of an API call can often not be determined statically. (ii) Deprecation in API specifications is not the only way to signal deprecation for web APIs.

We introduce a dynamic approach using tracing to detect calls to web APIs. Subsequently, we check the called APIs for deprecation using an API specification, response meta-data, or a knowledge base. This approach addresses both limitations of the detection with static analysis. We implement the approach and evaluate it on three projects, including client-server calls as well as a microservice benchmark system. The empirical evaluation yields a precision of 1.00 and a recall of 0.95. The false negatives can be attributed to a shortcoming in the automatic instrumentation provided by OpenTelemetry observability framework.

**Index Terms**—application programming interface, deprecation, dynamic analysis, tracing

### 1. INTRODUCTION

Software systems are typically not isolated units but interact with their surroundings [1]. Such communication with external systems and their interfaces may be a business requirement, thus posing an *architecturally significant requirement*, or a voluntary decision during the system's design, thus introducing the dependency as an *architectural constraint* for further development [2], [3]. In addition to systems external not belonging to the same organization, a similar setting is encountered within a system if it comprises highly decoupled modules, e.g., with a microservice architecture [4] or bounded contexts [5].

As software systems evolve, they also need to adapt their interfaces to changed functionality. A common way to inform clients calling these interfaces that their use is no longer encouraged is the *deprecation* of an entire API, an element in the API, or a particular version of the API. Depending on the further actions after a deprecation is introduced, it may lead to technical debt with both API clients [6] and API providers [7].

If clients wish to react to the deprecation of an API they depend on, they first need to be aware of the deprecation. While comprehensive support exists for detecting the deprecation of static APIs (e.g., for Java libraries) the situation is different for web APIs (e.g., for REST calls) [8].

Previous approaches for the detection of calls to deprecated web APIs have utilized static analysis methods similar to the case of static APIs [9]. However, this has two essential limitations: (i) The target of an API call can often not be determined statically. (ii) Deprecation in API specifications is not the only way to signal deprecation for web APIs. Thus, we introduce an approach to determine the usage of deprecated web APIs dynamically. To the best of our knowledge, it is the first such approach.

To overcome these limitations, we develop a new approach comprising two essential steps: (i) The execution of a client component is observed, and information about calls to web APIs is recorded. (ii) The recorded data is analyzed and each endpoint is checked for deprecation. Our approach considers deprecation that is signaled directly in the call's response, in an associated API specification, or in a knowledge base. We implement the approach for HTTP APIs and OpenAPI specifications. OpenAPI is a de-facto standard for the specification of REST APIs.

This paper's main contributions are:

- We present the first approach to identify the usage of deprecated web APIs dynamically.
- We implement the approach for REST APIs, OpenTelemetry data, and various forms of deprecation information.
- We evaluate the approach on multiple sample systems.
- We include a replication package with code, examples, and the evaluation data [10].

These contributions benefit practitioners who use or offer deprecated APIs. Furthermore, the contributions benefit researchers who wish to study the deprecation of web APIs.

Section II introduces fundamentals, motivates our research, and surveys related work. In Section IV, we introduce our approach and present its implementation. Subsequently, we evaluate the approach and its implementation in Section V and discuss our results and their limitations in Section VI. Finally, Section VII concludes the paper.

# Application Programming Interface (API)

**API** – interface to another software component or system using means of programming languages

**Local APIs** – interfaces to elements within the same ecosystem, usually within a programming language; statically resolved at build time  
e.g., Java libraries (via Maven, JAR files, ...)

**Remote APIs** – interfaces using means of network communication  
e.g., message-oriented middleware (Kafka, ...) or Web APIs (REST, WebSocket, ...)

# Deprecation of APIs



**deprecated**


the use of this element is discouraged

**Breaking changes:** changes in an API that potentially break existing client code

– violation of backward compatibility

**Example:** removal of a method

# Deprecation: Java APIs (local, static)

 The method `isSpace(char)` from the type `Character` is deprecated

```
boolean result = Character.isSpace('@');
```

`isSpace('@')`

`false`

```
@Deprecated  
public static boolean isSpace(char ch) {...};
```

## Deprecation: REST APIs (remote, dynamic)

```
HttpClient client = HttpClient.newBuilder().build();  
HttpRequest request = HttpRequest.newBuilder().  
    uri("https://character.com/isSpace").build();  
HttpResponse response = client.send(request, myBodyHandler);
```

POST /isSpace HTTP 1.1  
Host: character.com  
  
{ ... "character":"@", ... }

HTTP/1.1 200 OK  
Date: Thu, 27 Feb 2025  
  
{ ... "result":false, ... }



## Deprecation: REST APIs (remote, dynamic)



The API element  
*isSpace* is deprecated.

```
HttpClient client = HttpClient.newBuilder().build();
HttpRequest request = HttpRequest.newBuilder()
    .uri("https://character.com/isSpace").build();
HttpResponse response = client.send(request, myBodyHandler);
```

POST /isSpace HTTP 1.1  
Host: character.com  
  
{ ... "character":"@", ... }

HTTP/1.1 200 OK  
**Deprecation: Wed, 31 Dec 2025**  
  
{ ... "result":false, ... }



**deprecated: true**



# Related Work

Deprecation info is not always provided [Yasmin et al. 2020] [Di Lauro et al. 2022], and developers assume unlimited availability [Lercher et al. 2023].

Static approach in Javascript detects calls and checks deprecation [Yasmin 2021].

Static approaches to identify calls in Java

search for hard-coded URL strings [Rapoport 2017] [Gadient 2020]

or assume prior knowledge [Pigazzini et al. 2020] [Genfer & Zdun 2021].

# Challenges for Remote APIs

Calls to remote APIs are harder to detect and may use third-party libraries

- e. g. more than 15 popular HTTP clients in Java.

Call targets may not be present in the source code

- in particular, they may be determined only at runtime.

Remote APIs may be deprecated later

- even if no deprecation has been declared when the client is developed.

# Design Principles for our Dynamic Solution

- DP1** The identification should not be targeted to a single programming language.
- DP2** The identification should include calls with targets that are only known at runtime.
- DP3** The identification should include deprecation information from API specifications, HTTP header data in responses, and knowledge bases.
- DP4** No prior information about an API should be required for the identification.

# Tracing (OpenTelemetry)

- **Instrumentation:**

Instructing the application/runtime environment to record specified events.

- **Span:**

Individual record of an observed event.

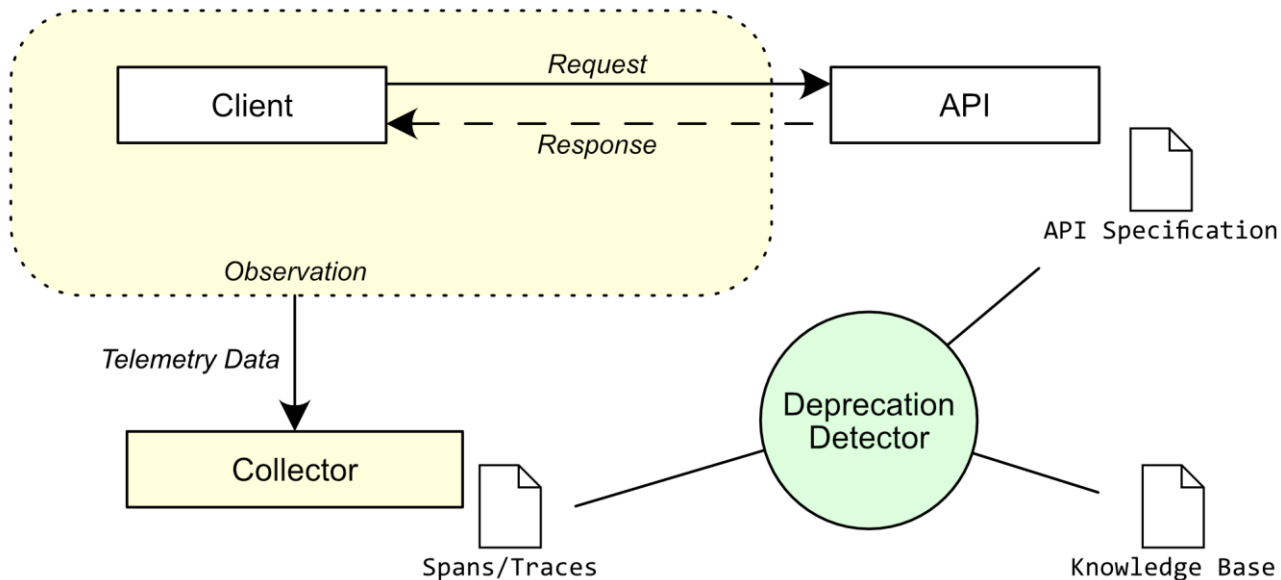
- **Trace:**

Set of related spans, covering one call – possibly through multiple components.

- **Metric:**

Measurement of runtime properties based on the observation.

# Our Approach with OpenTelemetry



# Example Observations

```
{  
  "key": "http.url",  
  "value": {  
    "stringValue": "https://character.com/isSpace?character=%40"  
  }  
}
```

*Request*

```
{  
  "key": "http.response.header.deprecation",  
  "value": {"arrayValue": {"values":  
    [{"stringValue": "Wed, 31 Dec 2025 23:59:59 GMT"}]  
  }}  
}
```

*Response*

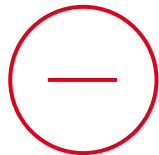
# Evaluation on Sample Projects



It works! 😊

It works well! 😊

- Precision 1.0
- Recall 0.95



- relies on instrumentation provided by OpenTelemetry
- only applicable dynamically (with typical disadvantages)
- only evaluated on simple sample projects

# Outlook

this  
approach

- evaluation in industrial settings
- usage without telemetry framework
- privacy / anonymization of URLs

general  
problem

- combination with static approaches
- improvements for API providers



# Summary

- Deprecation is well-researched and well-supported for local/static APIs.
- Static approaches on detection of deprecation fail for remote APIs.
- We present a dynamic approach for web APIs:
  - We monitor outgoing API calls.
  - We check traces for deprecation and cross-check with other sources.
- It works (on artificial projects).

# Overview

